Technical Report

CMU/SEI-93-TR-11
ESC-TR-93-188

# Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology

Robert W. Krut, Jr.

July 1993

DTIC
S ELECTE D
APR 1 8 1995
G

19950417 151

Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology
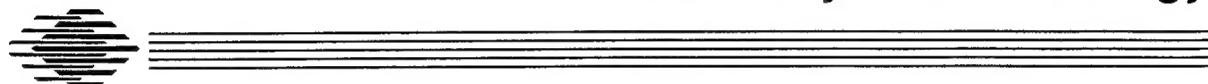
Robert W. Krut, Jr.

Application of Software Models Project

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESC/ENS
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official
DoD position. It is published in the interest of scientific and technical
information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

Table of Contents

List of Figures

List of Tables

Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology

Abstract: This report addresses the need for additional tool support for the Feature-Oriented Domain Analysis (FODA) methodology, developed at the Software Engineering Institute (SEI). Previous FODA studies relied on multiple tools to represent the components of a domain model. This report discusses the ability to represent an analyzed domain within the confines of a single support tool. This discussion was based on the transformation of a recently completed domain analysis from a multi-tool, multi-view representation into a single tool which represents the multiple views of a FODA domain model. This report also describes the potential for prototyping of systems using the FODA domain analysis products and the supporting tool.

1   Introduction

1.1   Background

In 1990, the Software Engineering Institute (SEI) introduced a domain analysis methodology known as Feature-Oriented Domain Analysis (FODA). The feature-oriented concept is based on the emphasis placed by the methodology on identifying those features a user or customer commonly expects in applications in a domain. The FODA method supports the discovery, analysis, and documentation of commonality and differences within a domain.

The application of the FODA method to a window management system was illustrated by the FODA feasibility study [SEI90a]. The feasibility study identified the need for tools to support both the process of domain analysis and the process by which the products of the domain analysis support software development. The initial intention of the feasibility study was to perform the analysis using manual techniques. As the amount of information needed to describe the domain grew, the manual technique became more complex. To handle the volume and complexity of information gathered during the feasibility study, a set of manual and independent semi-automated methods were used.

Representing the results of a domain analysis process is primarily a task of representing a large volume of information. The domain analyst should provide information so that the user of the analysis can access that knowledge quickly and easily. The goal of domain analysis tool support should be to offer an integrated environment for collecting and retrieving the domain model and architectures. The set of manual and independent semi-automated methods used during the feasibility study did not meet that goal. Therefore, the FODA feasibility study recommended that subsequent domain analysis studies investigate integrating tool support into the domain analysis method.

In 1991, the SEI continued to evolve and validate the FODA method by applying FODA to a more challenging domain. The Army Movement Control Domain was selected as representa-

tive of a larger, more complex, and less well-documented domain [SEI92a]. As part of this application, the recommendation to further integrate tool support into the domain analysis method was addressed. Domain information, initially captured by the manual and semi-automated methods from the feasibility study, was recaptured using a single tool to represent the multiple views of FODA. The support tool employed during the Movement Control Domain Analysis was 001™, created by Hamilton Technologies, Inc. (HTI) [HAMIL91], [MURPH90].

1.2 Purpose of Report

The purpose of this report is to document the findings of integrating 001 into the FODA method. The test bed for the integration was the application of the FODA method to the Movement Control Domain. The primary focus of the integration was the domain modeling phase of FODA.

This report addresses the ability of 001 to support both the process of domain analysis and the process by which the products of domain analysis support software development. Specifically, the discussion will focus on 001's ability to:

- Represent the domain model.
  - Represent entities, features, behavior, and functionality within the domain.
  - Integrate entities, features, behavior, and functionality into a consistent model.
- Generate code for application prototyping.
  - Generate an application prototyper from the domain model.
  - Map selected features to an application under development.

While the information contained in the domain model provides enough information to build a system, automatic prototyping gives the user the ability to validate the domain model, develop applications, and understand new capabilities created from a selection of features. Therefore, the ability of 001 to implement a working model of the system under development based on a selection of features becomes an extension of the previous FODA effort.

This report is the first of three related reports which document the current Domain Analysis activities at the SEI. The topics of the two remaining documents are "From FODA Models to Reusable Software: Guidelines for Architecture and Ada Design" and "Guidelines for Implementing Domain Analysis/Domain Engineering in Your Organization."

The intent of this report is not to imply that 001 is the only solution to the integration of a support tool into the FODA methodology. 001 offered the tools to represent and prototype the domain model. The selection of 001 to represent the FODA methodology was not the result of a long, in-depth study of support tools.

1.3 Audience for Report

The report is directed towards individuals generally interested and knowledgeable in the application of the FODA method or similar domain analysis methods. The intended audience need not be experts in the area of movement control or 001. The primary focus was to demonstrate one tool's ability to support the FODA method.

The information contained in this document provides a brief description of the FODA method and 001. However, this report was not designed to be a tutorial in either the FODA method [SEI90a] or 001 [HAMIL91].

1.4 Report Overview

This document contains the following major sections:

Section 2: Overview of Feature-Oriented Domain Analysis Method - This section reiterates both the foundations and principles of the FODA methodology and briefly outlines the phases of the methodology. Due to the intent of this report, the primary focus of this section was placed on the domain modeling phase and applying the results of domain analysis to a system under development. Detailed information on the FODA methodology is given in the FODA feasibility study [SEI90a] and the Application of FODA to the Army Movement Control Domain [SEI92a].

Section 3: Overview of Hamilton Technologies, Inc. 001 - This section introduces the components of the 001Tool Suite employed to support the FODA method. The in-depth languages and syntax of 001 are documented in the 001 Tool Suite System Reference Manual [001SRM].

Section 4: Representing the Domain Model - This section discusses how the products of the domain modeling phase of the FODA methodology were represented in the feasibility study and in the 001 Tool Suite. Information which could not be represented within the 001 Tool Suite were identified.

Section 5: Application of the Domain Model in System Development - This section discusses how the domain model products are used in the construction, usage, and validation of a prototype for modeling systems in the domain. The discussion extends into the ability of 001 to generate code from the 001-represented domain model for application prototyping.

Section 6: Conclusions - This section discusses the outcome of integrating the 001 technology into the FODA methodology. The discussion covers the advantages and limitations of the 001 Tool Suite in representing the products of FODA as well as recommendations for enhancements in future 001 releases. The section concludes with a discussion of expanding the role of tool support in future FODA projects to enhance the presentation of the products of FODA.

2 Overview of Feature-Oriented Domain Analysis Method

The Feature-Oriented Domain Analysis (FODA) methodology resulted from an in-depth study of other domain analysis approaches [SEI90a]. Successful applications of various methodologies pointed towards those approaches which focused on the process and products of domain analysis. As a result, the FODA feasibility study established methods for performing a domain analysis, described the products of the domain analysis process, and established the means to use these products for application development. The feature-oriented concept of FODA is based on the emphasis placed by the method on identifying prominent or distinctive *features* within a class of related software systems. These features lead to the creation of a set of products that define the domain.

This section reiterates the foundation of the FODA concepts and provides an overview of each of the phases within the FODA process and the relationships between their products and their consumers.

2.1 Foundations of the FODA Methodology

The FODA methodology was founded on a set of modeling concepts and primitives. These concepts and principles are used to develop domain products that are generic and widely applicable within a domain.

The basic modeling concepts used in the creation of the domain products are *abstraction* and *refinement*. Abstraction is used to create domain products from the specific applications in the domain. These generic domain products abstract the functionalities and designs of the applications in a domain. The generic nature of the domain products is created by abstracting away "factors" that make one application different from other related applications. The FODA method advocates that applications in the domain should be abstracted to the level where no differences exist between the applications.

Refinements are used to both refine the generic domain products and to refine the domain products into applications. Once the abstraction of the applications in the application domain is completed, the factors that make each application unique are incorporated into the generic domain products as refinements of the abstractions. Specific applications in a domain may be developed as further refinements of the domain products by using the general abstraction as a baseline and selecting among alternatives and options to develop the application (i.e., those factors that have been abstracted away must be made specific and reintroduced).

Abstracting the applications in the application domain is accomplished by using the modeling primitives of: aggregation/decomposition, generalization/specialization, and parameterization. The FODA method applies the aggregation and generalization primitives to capture the commonalities of the applications in the domain in terms of abstractions. Differences between applications are captured in refinements. An abstraction can usually be refined (i.e.,

decomposed or specialized) in many different ways depending on the context in which the refinements are made. Parameters are defined to uniquely specify the context for each specific refinement. The result of this approach is a domain product consisting of a collection of abstractions and a series of refinements of each abstraction with parameterization. Understanding what differentiates applications in a domain is most critical since it is the basis for abstractions, refinements, and parameterization.

Domain products are produced through a number of activities. The following subsection discusses the activities of the FODA process and the models that are produced from the process.

2.2 FODA Process and Products

The FODA feasibility study [SEI90a] defined a process for domain analysis and established specific products for later use. Three basic phases characterize the FODA process:

1. *Context Analysis*: defining the extent (or bounds) of a domain for analysis

2. *Domain Modeling*: providing a description of the problem space in the domain that is addressed by software

3. *Architecture Modeling*: creating the software architecture(s) for implementing solutions to the problems in the domain

Figure 2-1 provides the structure of the FODA methodology and Table 2-1 summarizes the inputs, activities, and products of each phase in the FODA process and the relationships between their products. A textual overview of each of the phases is given in the following subsections.
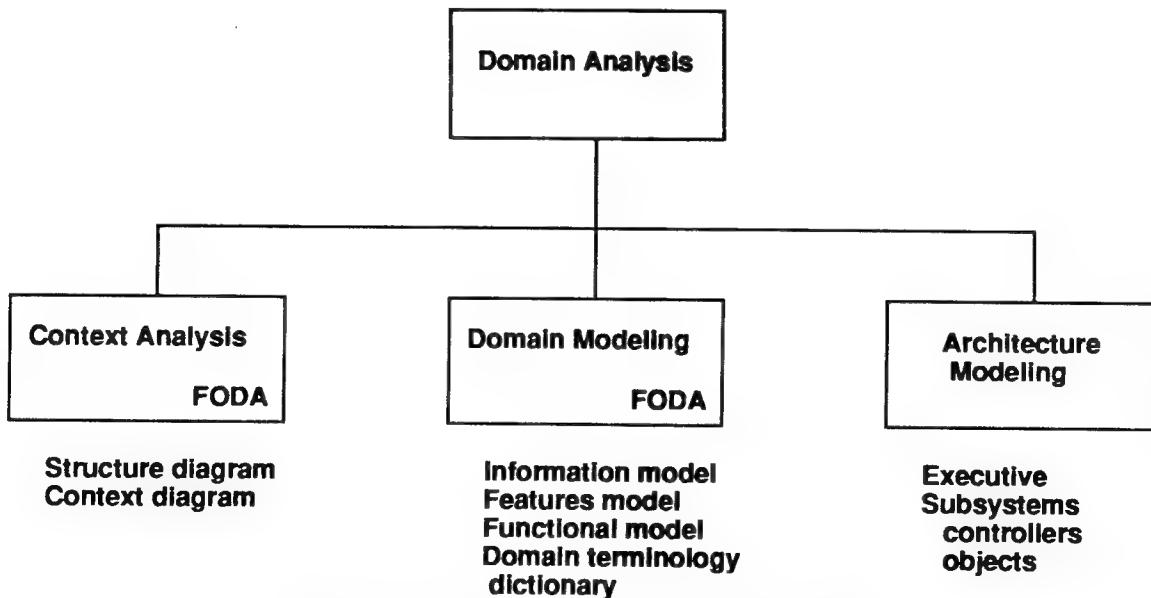
Figure 2-1 FODA Methodology Structure

Phase	Inputs	Activities	Products
Context Analysis	Operating environments, Standards	Context analysis	Context model
Domain Modeling	Features, Context model	Features analysis	Features model
	Application domain knowledge	Information modeling	Information model
	Domain technology, Context model, Features model, Information model, Requirements	Functional analysis	Functional model
			Behavioral model
Architectural Modeling	Implementation technology, Context model, Features model, Information model, Design information	Architectural modeling	Structured executive
			Subsystems model(s)

Table 2-1 Summary of the FODA Method

2.2.1 Context Analysis

Context analysis defines the scope of a domain that is likely to yield useful domain products. During the context analysis of a domain, the relationships between the "domain of interest" and the elements external to it are established and analyzed for variability. The kinds of variability to be accounted for are, for example, when applications in the domain have different data requirements and/or operating environments. The results of the context analysis, along with other factors such as availability of domain expertise, domain data, and project constraints, are used to limit the scope of the domain.

The product resulting from the context analysis is the *context model.* This model includes a structure diagram and a context diagram. The structure diagram for this domain is an informal block diagram in which the domain is placed relative to higher-, lower-, and peer-level domains. Higher-level domains are those of which the domain under analysis is a part or to which it applies. Lower-level domains (or subdomains) are those within the scope of the domain under analysis, but which are well understood. Any other relevant domains (i.e., peer domains) must also be included in the diagram.

The context diagram is a data flow diagram showing data flows between a generalized application within the domain and the other entities and abstractions with which it communicates. One thing that differentiates the use of data flow diagrams in domain analysis from other typical uses is that the variability of the data flows across the domain boundary must be accounted for with either a set of diagrams or text describing the differences.

These products provide the domain analysis participants with a common understanding of:

- The scope of the domain
- The relationship to other domains
- The inputs/outputs
- Stored data requirements (at a high level) for the domain

2.2.2 Domain Modeling

Domain modeling identifies the commonalities and differences that characterize the applications within the domain. The domain modeling phase consists of three major activities. A brief description of each activity and its results is given below.

1. *Information Modeling* captures and defines the domain knowledge and data requirements that are essential for implementing applications in the domain. Domain knowledge typically is information that is deeply embedded in the software and is often difficult to trace. Those who maintain or reuse software need this information in order to understand the problems the domain addresses.

   The information model may take the form of an entity-relationship (ER) model [SEI90a], a semantic network [SEI92a], or other representations such as object modeling [RUMB91].

   The information model is used primarily by the requirements analyst and the software designer to ensure that the proper data abstractions and decompositions are used in the development of the system. The information model also defines data that is assumed to come from external sources.

2. *Features Analysis* captures a customer's or end user's understanding of the general capabilities of applications in a domain[1]. For a domain, the commonalities and differences among related systems of interest were designated as features and are depicted in the *features model*. Thesefeatures, which describe the context of domain applications, the needed operations and their attributes, and representation variations are important results because the features model generalizes and parameterizes the other models produced in this domain analysis.

---

[1] A user may be a human user or another system with which applications in a domain typically interact.

The features model is the chief means of communication between the customers and the developers of new applications. The features are meaningful to the end users and can assist the requirements analysts in the derivation of a system specification that will provide the desired capabilities. The features model provides them with a complete and consistent view of the domain.

3. *Functional Analysis* identifies the control and data flow commonalities and differences of the applications in a domain. This activity abstracts and then structures the common functions found in the domain and the sequencing of those actions into a model. Common features and information model entities form the basis for the abstract functional model. The control and data flow of an individual application can be instantiated or derived from the functional model with appropriate adaptation.

The functional model is the foundation upon which the software designer begins the process of understanding how to provide the features and make use of the entities selected.

The domain modeling process also produces an extensive *Domain Dictionary* of terms and/or abbreviations that are used in describing the features and entities in the model and a textual description of the features and entities themselves.

The domain dictionary has been found to be one of the most useful products of a domain analysis. The dictionary helps to alleviate a great deal of miscommunication by providing the domain information users with:

- a central location to look for terms and abbreviations that are completely new to them

- definitions of terms that are used differently or in a very specific way within the domain

2.2.3 Architectural Modeling

Architectural modeling provides a software solution for applications in the domain. An architectural model (also known as a design reference model) is developed in this phase and detailed design and component construction can be done from this model. This architectural model is a high-level design for applications in a domain. It focuses on identifying concurrent processes and domain-oriented common modules and on allocating the features, functions, and data objects defined in the domain models to the processes and modules.

2.2.4 Applying the Results of Domain Analysis

FODA defines a method for performing domain analysis and describes the products of an analysis.

Figure 2-2 shows the three components of the domain model: the features model, the information model, and the functional model. A system developer works with the domain analyst and these products to define requirements for a system. The three steps in the process are:

1. The developer and domain analyst use the features model as a vehicle for communicating system needs. The domain analyst will turn these needs into a selection of features. In addition, composition rules among features will automatically add specific features to the new system.

2. The domain analyst uses the information model to explain the *objects* that comprise a system. This helps the system developer understand the data requirements as well as other systems and data structures with which the system must interoperate.

3. The functional model is then used to describe commonality and differences in data and control flow resulting from differing combinations of features.

The product of feature selection is the definition of capabilities of the system under development as shown in Figure 2-2.
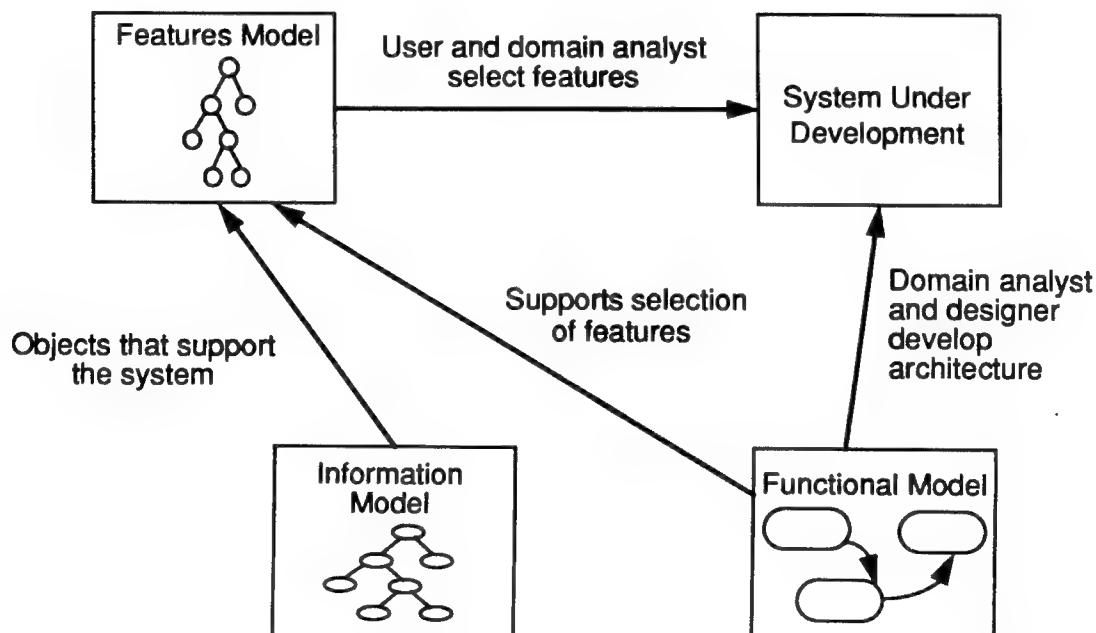
Figure 2-2 Use of Domain Model in System Development

The functional model supports feature selection as well as architectural development. Feature selection will parameterize the functional model, establishing the dynamics of interacting system capabilities. A system developer will utilize this information in making choices that will affect both system control and operations. For example, a choice of features may affect the sequence of operations or eliminate those operations altogether. Another important aspect of this model is the definition of data flow resulting from these operations. The system dynamics necessary to meet the desired system capabilities may depend on specific feature selections.

When implementing the desired features, the domain analyst and software designer will work jointly to establish the software architecture. The functional model defines data exported by specific activities as well as those required for input by other activities. The model also shows the control necessary to start an activity to effect the data flow. The detailed realization of all data flows and the control necessary to accomplish them are key components of software design. Using the features model, the software designer engineers a general architecture that supports implementation of common features that can also be parameterized for tailorability for meeting specific optional and alternative features.

Domain analysis and its products are the model base for understanding user needs and obtaining requirements. Where the models are inadequate for understanding the problem space and producing a solution, the life cycle must include unprecedented development. In addition to filling the gaps in existing models, unprecedented development will lead to refinement of existing models or to the realization that the existing models are no longer adequate.

Chapter 5 of this report further discusses the application of the domain model in system development.

3 Overview of Hamilton Technologies, Inc. (HTI) 001

001 is a technology which has been designed, developed, and used for the rapid development of systems. The 001 technology embodies many aspects of a "Development Before the Fact Approach" [HAMIL91] which focuses on developing systems with built-in quality and productivity. The 001 technology is based on a set of axioms [HAMIL91] that verifies consistency and logical completeness of the resulting system design. All aspects of the system design are expressed in the same "language," from the highest level concepts to the most detailed implementation specifics. The following subsections discuss the 001 Tool Suite which ensures the correct usage of the 001 technology and the 001 systems development process.

3.1 The 001 Tool Suite

The 001 Tool Suite is an integrated family of automated software tools designed to improve the system development process. The tool suite automates the application of the 001 philosophy to fully integrate data structures, object design, and functional performance.

The 001 Tool Suite [001SRM], summarized in the next paragraphs, consists of the following components:

- The 001 AXES language
- A textual editor
- A graphics Map Editor (MapE)
- The Analyzer
- A Resource Allocation Tool (RAT)
  - a function-oriented RAT
  - a type-oriented RAT
- The systems management interface (FACE)
- An Object Map editor (OMap)

The 001 AXES language provides a means of integrating cross-checking and consistency with reuse of data and features. The language includes an object-type decomposition via a Type Map (TMap) and functional decomposition via a set of Functional Maps (FMaps). The TMap defines the possible objects and the states that an object may have. FMaps are used to define and control the transformation of objects from one state to another state. A Road Map (RMap) graphically provides the hierarchical "table of contents" of FMaps and TMap and manages access to the FMap and TMap definitions.

Both the graphical and textual editors are used to construct an 001 AXES definition. The graphical map editor is used to graphically construct and edit FMaps, the TMap, and the

RMap. The textual editor is used to create textual definitions of FMaps or TMap, or to access the textual version of the graphical definitions maintained by MapE.

The graphical TMap consists of a set of trees. Each tree represents the decomposition of an object. The syntax of a TMap provides four abstract types to represent the decomposition of an object into its component objects. These are the *TupleOf*, *OneOf*, *OsetOf*, and *TreeOf* abstract types.

- The *TupleOf* abstract type identifies an object as consisting of one to a specified number of component objects.

- The *OneOf* abstract type identifies an object as being one of its component objects (i.e.,the object instance may be represented by one and only one of its component objects).

- The *OsetOf* abstract type represents an object as being an ordered set of its component objects. This is similar to the construct of a circular, doubly-linked list available in some programming languages.

- The *TreeOf* abstract type is used to represent an arbitrary tree structure with an object at each node.

In the graphical FMaps, the 001 user specifies a particular functionality as a tree of functions, with each function specifying its inputs and outputs. In addition, with each function there is an associated control structure specifying constraints on the way that data (inputs and outputs) may flow between the functions that make up that function's decomposition.

The Analyzer performs the syntax and semantic analysis on partial or completed definitions produced by either the textual editor or MapE. The Analyzer checks to ensure that all parts of the definition are internally consistent and checks all interfaces for correctness and completeness.

The RAT generates operational code. A function-oriented RAT generates a target language source code program from successfully analyzed definitions. It ensures that the implementation maintains the integrity of its 001AXES definition. It eliminates error-prone hand-coding, permits simulation, and makes rapid prototyping possible. A type-oriented RAT generates abstract type templates used by the functional RAT. These type templates define the allowed primitive operations on each type.

The systems management interface is designed to allow easy access to all of the capabilities of 001 and a wide variety of general purpose commands to be executed.

An OMap can be thought of as the runtime instance of data that has been created and organized according to the constraints provided by a particular TMap. The OMap editor is a tool that allows the user to readily access such data and manipulate it in a variety of powerful ways. For example, the Omap editor can be used as a default-form user interface for the system during execution.

3.2 The 001 Development Process

The 001 systems development process consists of four phases: defining, analyzing, generating, and executing. First, a model of the system is defined using FMaps and TMap. The FMaps and TMap combined form an integrated description of the system. Next, the FMap and TMap definitions are analyzed via the 001 Analyzer to ensure that the model was defined properly. The RAT is then used to automatically generate a software implementation that is consistent with the model. The resulting source code can be compiled and executed. This executable model represents the prototype of the system.

4   Representing the Domain Model

In this study, the domain data was initially represented by the semi-automated methods from the feasibility study. The domain data was then represented by the 001 AXES language using the TMap to model both the entities and features of the FODA method[1]. The FMaps were used to define the functionality and behavior of the system in terms of functional decomposition, control structures, and the flow of data.

This section presents the highlights of representing the FODA domain model using the 001 AXES. Each subsection briefly describes a modeling activity (i.e., information modeling, features modeling, functional modeling) within the domain modeling phase. The automated support used during the feasibility study and the corresponding 001 representation of the activity are discussed. The representations employed during the feasibility study are referred to as the baseline representation. Figure 4-1 lists the representations used for each of the FODA domain modeling phases.

Phase \ Representation	Baseline	001
Information Modeling	Chen's techniques Entity-relationship diagrams Automated drawing tool	TMap
Features Modeling	Structure diagrams Automated drawing tool Prolog	TMap
Functional Modeling	Statemate	FMap

Figure 4-1 Representations vs. FODA Modeling Phases

The originally developed Movement Control Domain Model (with notation seen in the FODA feasibility study) and the 001-represented Movement Control Domain Model are not provided in this discussion due to the amount of information represented by the domain model. However, each subsection will provide a sample of each of the representations from the Movement

---

[1] In both representations, the information model was represented by an entity-relationship model.

Control Domain Model[1]. A more complete set of representations of the models and definitions of the nodes are provided in References [SEI92a].

Information contained within the domain dictionary is discussed within each subsection along with the ability to integrate this information into the 001 representations.

Validation of the resulting models was performed to determine whether the applications within the domain were properly represented. In the feasibility study, validation was performed by visually inspecting the components of the entity-relationship model for completeness whereas automated support was employed for validating the features and functional models. Validation of the 001 representations was performed by using the definitions analyzer and the prototyping capability of 001. This section discusses the automated support used to validate the features and functional models from the feasibility study. Validation of the TMap and FMaps are briefly discussed in this section. Additional validation and prototyping of the 001 representation of the domain model will be addressed in Section 5.

4.1   Information Model

In FODA, information modeling is used to capture and define the domain knowledge that is essential for implementing applications in the domain. The information model supports analysis and understanding of domain problems and assists in the derivation and structuring of domain objects. When the information model is represented by an entity-relationship (ER) model, entities can be used to identify domain objects, which are then used to define data flows and data stores in the functional model.

The ER model represents domain knowledge explicitly in terms of domain entities and their relationships, where an entity is either a physical entity or a concept. The ER model used in this phase of FODA consists of three parts:

1. Entity-relationship diagram

2. Attributes of the entities

3. Constraints on the entities and relationships

The next two subsections discuss the baseline and 001 representations of the ER model.

4.1.1   Baseline Representation of the Entity-Relationship Model

The ER modeling technique in the feasibility study was an adaptation of Chen's method and semantic data modeling [SEI90a]. The basic building blocks of the ER models were entity classes and the generalization and aggregation concepts from semantic data modeling. Ag-

---

[1] These samples are provided to give the reader a sense of what the representations look like although the information within the representations may not have a one-to-one correspondence.

gregation relationships specify composition structures between entities while generalization relationships specify commonalities and differences among entities.

During the feasibility study, it was recognized that the high-level representation of an ER model was sufficient for an overview of the entities in the domain. However, it became apparent that an automated ER modeling tool would be required in order to manipulate more detailed ER data and maintain completeness and consistency. No modeling tools that support the FODA approach to ER modeling, which combines ER modeling with semantic data modeling, were found at the time of the feasibility study. Therefore, graphical representations of the ER diagrams were created using a basic drawing tool following Chen's diagrammatic technique for exhibiting entities and relationships. Entity classes were used to represent abstractions of objects in the application domain and the relationship types *is-a* and *consists-of* represented generalization and aggregation relations, respectively. In additional, relationship types other than the *is-a* and *consists-of* that are important for the domain were defined and used as part of the ER model. For example, a *has* relationship was defined to represent a one-to-many, parent-child, relationship. Figure 4-2 provides an example of the baseline ER model.
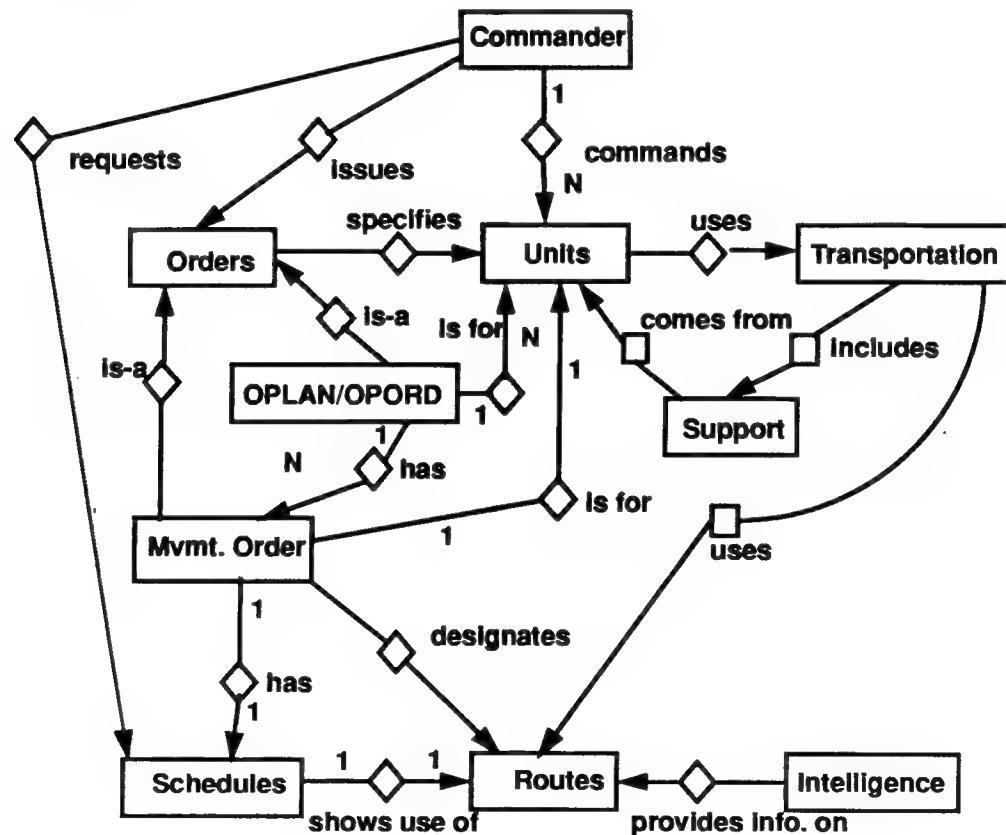

Figure 4-2 Example of the Baseline ER Model

The attributes of an entity and constraints on the entities and relationships were captured as part of the domain dictionary. The domain dictionary was created using the Info mode of the

GNU Emacs editor. Textual definitions were created in a "forms-like" fashion, which enabled a user to browse entities, attributes, relationship types, and constraints for all of the elements within the ER model.

4.1.2   001 Representation of the Information Model

For the 001 implementation, the relationships between entities on the ER diagram were transformed into the relationships between object entities on an 001 TMap. The TMap provides a tree-like structure with each node corresponding to an object type. The TMap enables the modeling of the decomposition of objects using sets, arrays, trees, classification, reference, extension, and primitive types. Concepts of generalization, aggregation, and attributes were transformed using the TupleOf, OSetOf, and OneOf abstract types within the 001 TMap syntax. The TupleOf abstract type was used to decompose a parent object into different component parts (children types). These component parts may be objects of the same type or objects of different types. The one-to-many, parent-child relationship was represented with the OsetOf abstract type. The OSetOf abstract type represents an ordered set of objects, containing zero or more objects of the same type. The OneOf abstract type was used to represent entities (or objects) in which there are many possible children types yet, when an object instance is created, exactly one of the child types exist. This abstract type was beneficial when addressing the instances of a member of an abstraction class which are a restriction of the class underlying the abstraction. Figure 4-3 provides an example of the OO1 information model.
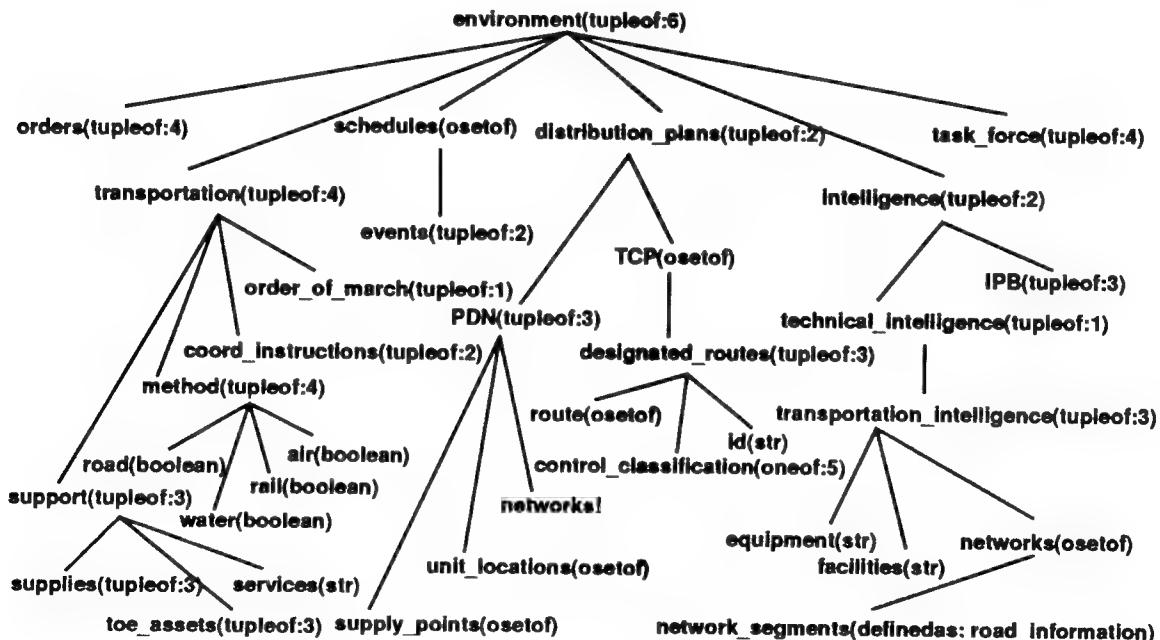

Figure 4-3 Example of an OO1 Information Model

Validation of the 001-represented information model consisted of a visual inspection of the components of the model for completeness and a definitions analysis from the 001 Analyzer.

The analyzer performed syntax and semantic analysis within the TMap definitions to check all interfaces for correctness and completeness to ensure internal consistency.

The domain dictionary for the 001 representation was again created by using the Info mode of the GNU Emacs editor. The TMap notation did not provide a facility to "tag" a definition to an entity for documentation (although this ability does exist for the FMap operations).

4.2 Features Model

In the FODA methodology, the purpose of features analysis is to capture in a model a customer's or end user's understanding of the general capabilities of applications in a domain. The features model represents the set of commonalities and differences of application capabilities within the domain. These capabilities from the perspective of end users are modeled as features.

The key elements within the features model are:

- A structure diagram (i.e., a tree-like diagram) containing a hierarchical decomposition of features and an indication of whether or not each feature is common (i.e., mandatory), alternative, or optional
- A definition for each feature in the model and an indication whether the feature should be "bound" (i.e., fix the value of a feature) at compile time, activation time, or at runtime
- Composition rules for features
- A record of issues and decisions

Composition rules define the semantics existing between features that are not expressed in the features diagram. Composition rules have two forms: (1) one feature requires the existence of another feature, and (2) one feature is mutually exclusive of another.

Issues and decisions are any factors (other than features) that cause functional differences between the applications. For example, issues and decisions may capture the rationale behind the selection of options and alternatives.

The next two subsections discuss the baseline and 001 representations of the features model.

4.2.1 Baseline Representation of the Features Model

In the feasibility study, a generic drawing tool was used to create the structure diagrams for the features model. These features diagrams were represented by an and/or tree of different features and a particular drawing style to show the relations among the features. The structural relationship *consists of* was used to represent a logical grouping of features. A line drawn between a child feature and a parent feature indicated that the child requires the parent to be present. If the parent was not marked as valid, then the child feature for the system was in essence "unreachable." Alternative and optional features of each grouping were indicated in

the features diagrams by joining alternative features with arcs and labeling optional features with circles. Figure 4-4 provides an example of a baseline features model.
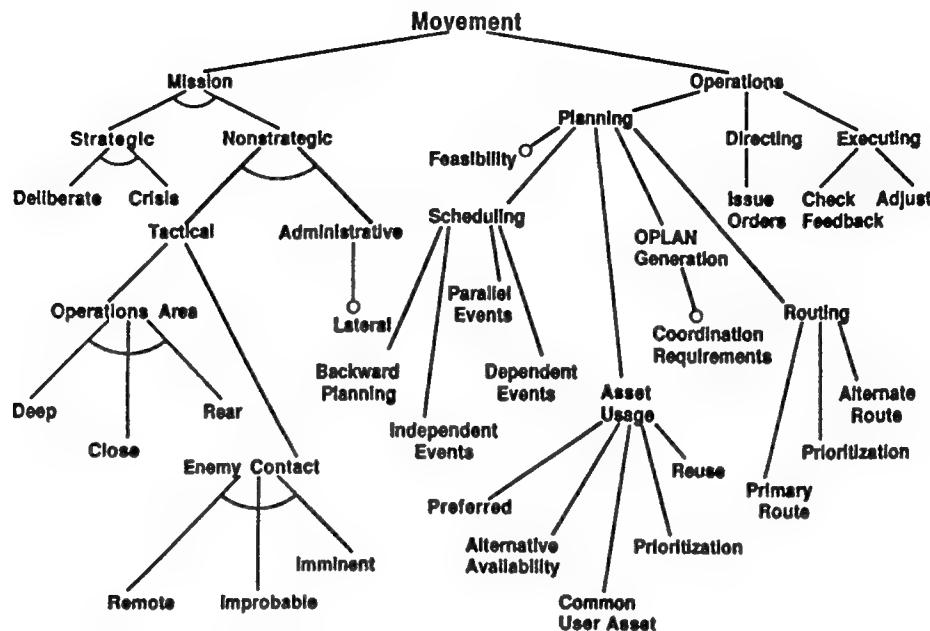
Figure 4-4 Example of a Baseline Features Model

Textual definitions of features, composition rules, and issues and decisions were created and stored as part of the domain dictionary. The Info mode of the GNU Emacs editor enabled browsing of features, composition rules, and issues and decisions for all of the features within the features model.

To validate the features model, a prototype tool was developed using Prolog. This tool enabled the features model developer to determine if the features model correctly represented the features of the domain. In this tool, the features were stored in a Prolog fact base, along with the composition rules and other related information. The tool enabled a user to define an existing or proposed system by allowing an arbitrary set of feature values to be specified and checked. Therefore, given a set of user-specified (i.e., "marked") features, the tool would perform the following functions:

- Check for all features that are specified, but which may not be reachable.
- Mark the features as "valid" if it is either:
  - marked "valid,"
  - mandatory,
  - not marked "invalid," or
  - *required* by a "valid" feature.

- Mark a feature as "invalid" if it is mutually exclusive with a "valid" feature.
- Produce an error if a feature is marked as both "valid" and "invalid."
- Enforce the proper selection of alternatives:
  - at least one alternative must be marked "valid."
  - more than one alternative cannot be "valid."

By selecting features from an existing application, the tool enabled the developer to determine if the features model correctly represented the features in that application. By repeating this procedure for all of the applications in the domain, the features model for the domain could be verified. Validation of the features model included the selection of features for a non-existent application(s) to determine the generality and applicability of the model when attempting to develop new applications.

4.2.2   001 Representation of the Features Model

For the 001 implementation, the features diagram was developed as a hierarchical decomposition of object features using the 001TMap. The features were identified and structured as optional, alternative, or mandatory by modeling the optional features as leaf-node objects of type *boolean* or *literal*, alternative features as a *OneOf* abstract type, and mandatory features as a *TupleOf* abstract type. Since a TMap follows the same tree-like structure as the baseline features diagram, the concept of "reachability" defined in the FODA feasibility study was maintained within a TMap. Figure 4-5 provides an example representation of a OO1 features model.The composition rules for features were not able to be represented within the TMap. This link between features was established within the FMap definitions as a set of checks on instantiations of the features which are bound by composition rules. In this manner, a feature would be instantiated as valid or invalid based on the selection of the user-specified features.

Validation of the features model was based upon the syntax checks within the TMap analyzer and the selection of features during execution of the prototype created from the domain model. Section 5 discusses in detail the execution of the prototype and the validation process of features. This discussion centers around the concept that the features model was created from known applications within the domain. Since the features in the features model are used to generalize and parameterize other models, the features model may be used to predict behavior in a given scenario based on the feature values of a specific application. The results of having two (or more) specific applications perform an operation may be compared with the results predicted by the features model instantiations for those applications. Any variation between the predicted and actual results should indicate problems with the descriptions of one or both applications.

As discussed in section 4.1.2, the domain dictionary for the 001 representation was created by using Info mode of the GNU Emacs editor since the TMap notation did not provide an facility to "tag" a definition to a feature for documentation.
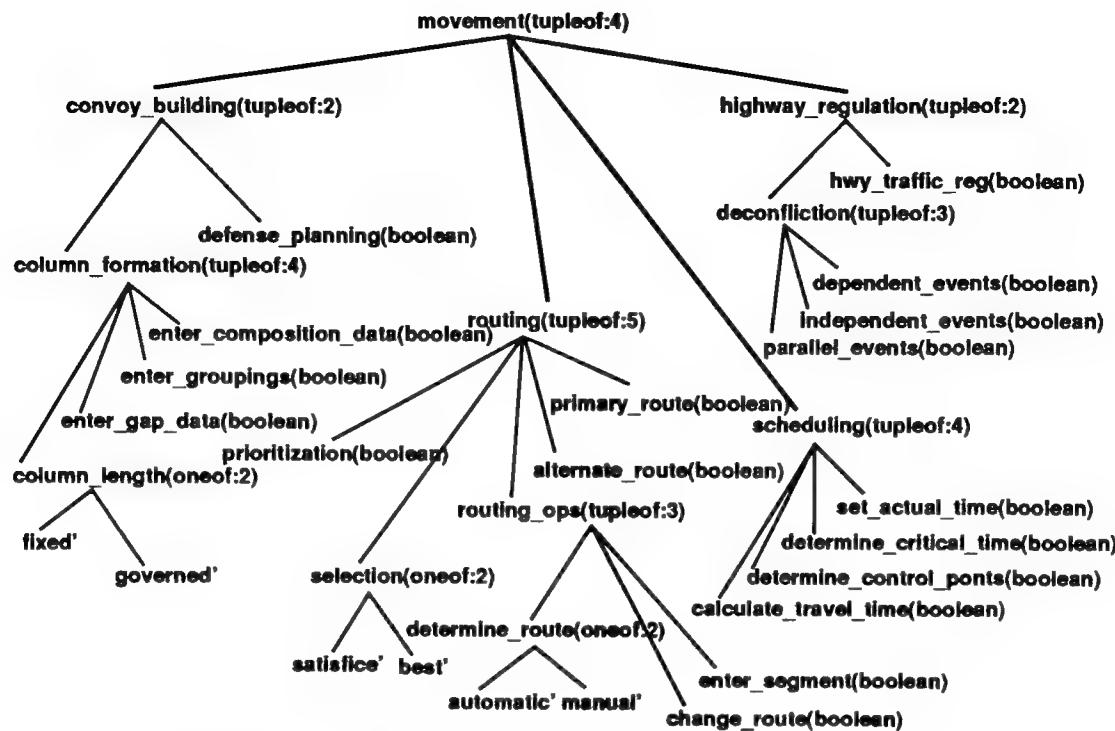
Figure 4-5 Example Representation of an OO1 Features Model

4.3 Functional Model

The functional model of the domain analysis identifies state and operational commonalities of the applications in a domain. The model also seeks to identify and compare differences between related applications. The functional model abstracts and represents these common/differing functions so that a specific application can be viewed as an adaptation or refinement of the model.

The features model and information model are used as guidelines in developing the functional model. A high-level, abstract functional model defines the operational characteristics of the mandatory features that operate on entities/objects. As the abstract functional model is refined, alternative and optional features are embedded into the model. Any issues/decisions raised during the features analysis are also incorporated into the model as refinements for parameterization. The resulting functional model represents the functionality of applications from an abstract level down to the detailed level.

Specifications of a functional model can be classified into two major categories: specification of functions and specification of behaviors. The specification of functions describes the structural aspect of an application in terms of inputs, outputs, activities, internal data, logical structures of these, and data-flow relationships between them. The specification of behaviors describes how an application behaves in terms of events, inputs, states, conditions, and state transitions.

The next two subsections discuss the baseline and 001 representations of the functional and behavioral aspects of the functional model.

4.3.1   Baseline Representation of the Functional Model

The feasibility study employed a commercially available automated system, Statemate [STA1],[STA2],[HAREL89], to represent and simulate the functional model. Statemate Activity-charts and Statecharts were used to represent the functional and the behavioral aspects, respectively. Figure 4-6 provides an example of an Activity-Chart and Figure 4-7 provides the corresponding Statemate StateChart.
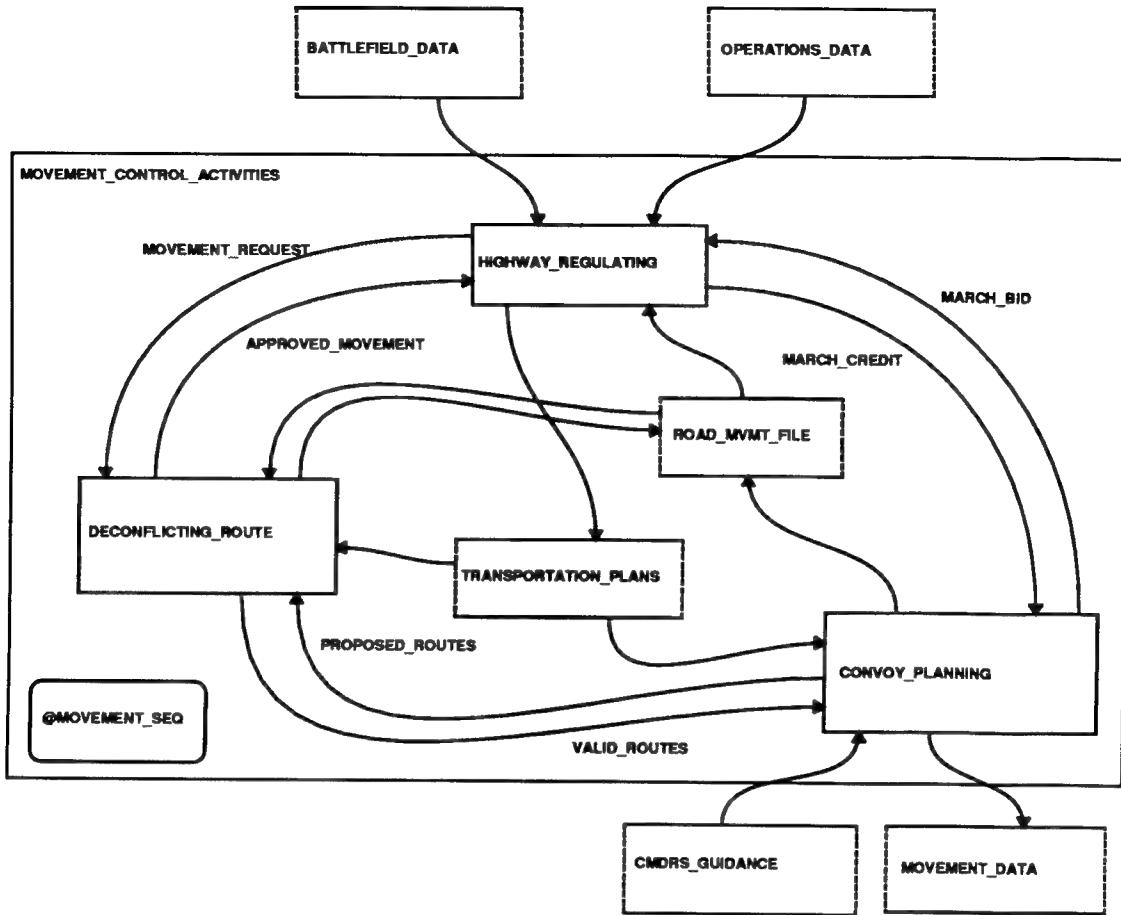
Figure 4-6 Example of a Statemate Activity-Chart

Activity-Charts follow the basic data flow techniques for representing functions (or activities) and the information that flows between them. Activity-Charts enable the overall functionality to be represented as a hierarchical decompositions of functions. The high-level functions represent the core functionality while the lower-level functions represent the differences among the applications.

Activity-Charts do not attempt to represent dynamic or behavioral issues. StateCharts are used to represent the behavior of an entire system or of a particular function within a system. StateCharts define the state and modes that the system might reside in and the transitions between them.
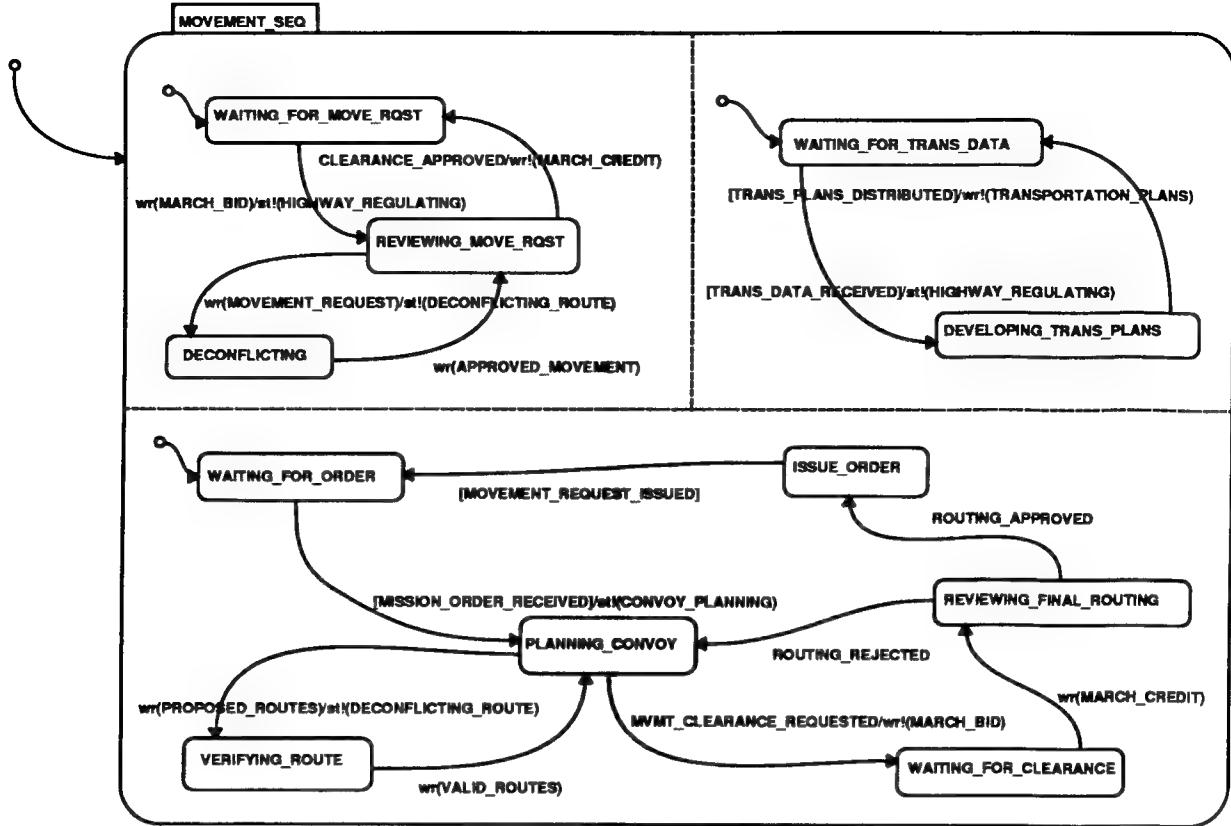
Figure 4-7 Example of a Statemate StateChart

No existing tool could be identified which adequately handled the modeling of common functionality and parameterization through features. Statemate offered a good, general-purpose specification and documentation tool, though the application of the tool to support domain analysis required tailoring. Through tailoring, Statemate could:

- Capture commonality
  - StateCharts show all states and transitions for specifying a behavioral view.
  - Activity-Charts show common functions and data flows (input and output) for specifying a functional view.
- Parameterize differences through features
  - StateCharts show alternative/optional features as conditions for modifying behavior.
  - Activity-Charts show optional data flow and provide textual descriptions.

The goal of validating the functional model is to verify that the model could be used to represent the performance of new or existing systems. To perform this verification, the functional model is refined using features of a specific application. The refinements entail parameteriza-

tion through feature selection to account for differences in the behavioral and functional views of the known applications. The feasibility study employed the simulation capabilities of Statemate to evaluate the performance of the baseline functional model. Statemate *conditions* were used to parameterize the specifications.

The Statemate simulation capabilities enable a user to visually walk through the functional and behavioral diagrams (i.e., Activity-Charts and StateCharts) of the functional model. Based on the declaration of the Statemate *conditions*, the simulation would step through the states and transitions of the behavioral view as well as the activities and data flows performed in response to these states and transitions. However, the simulation capabilities did not adequately support parameterization of the functional model. Statemate *conditions* for feature parameterization could not be distinguished from other *conditions* used in the specification.

4.3.2 001 Representation of the Functional Model

In the 001 representation, the functional model was represented by an RMap and a series of FMaps. The Rmap provided a hierarchical view of the functional components (subsystems and their decomposition derived from features) and the FMaps defined the functionality of each component of the decomposition.

An important aspect of a functional model is the ability to specify functions and behavior. Basic data flow and state transition diagrams (e.g., Statemate StateCharts and Activity-Charts) provide the least complex method for presenting this information. Although the 001 AXES are able to represent this information, the FMaps are more difficult to comprehend due to the syntax and semantics of the FMap language and the integration with the TMap. Therefore, the RMap was used to initially represent the top-most functional view as a hierarchical view of the functional components. The behavioral and functional aspects of these functional components are embedded in the FMaps.

The initial intention of the RMap is to provide an interface to manage the TMap and FMap definitions. However, by carefully developing the RMap, it may be used to represent a hierarchical view of the functionality from the abstract to the detailed functional component level.
Figure 4-8 represents the RMap from the Movement Control Domain Model. The RMap is a graphical, tree-like representation of the FMap hierarchies and dependencies. Each node on the RMap corresponds to an FMap name (or definition). By selecting FMap names which represent the underlying functionality of each FMap, the functional components of the domain are defined. The existing FMap hierarchies and dependencies defined the sequence in which the functions are performed. In combination, the FMap naming conventions, hierarchies, and dependencies enable the RMap to be viewed as the hierarchical decomposition of functionality in the domain. Higher-level functional components represent the functional commonalities of

the applications in the domain. Lower-level functional components represent the functional differences between the related applications.
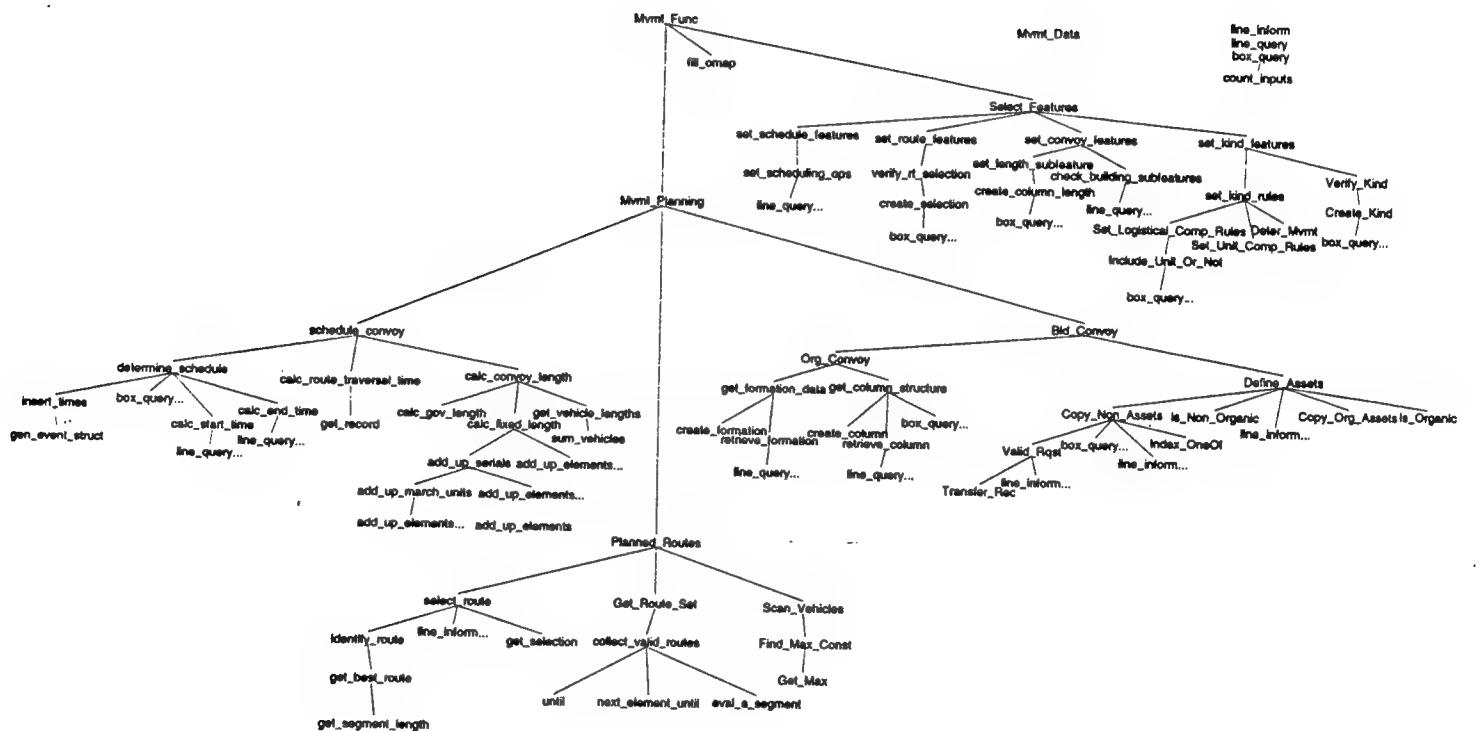
Figure 4-8 Movement Control Domain Model OO1 RMap

The FMaps were used to represent the specific functionality of each functional component. Each FMap is defined by a set of operations used to perform a specific function using OO1 AXES as the program design language. Operations define the behavior of the functional components in terms of functional decomposition, control structures, and the flow of data. Because the TMap and FMaps are integrated in 001, FMap definitions are developed to manage and manipulate the objects defined in the TMap. Therefore, FMaps define the functionality to be performed on the entities of the domain. In addition, FMaps enable the functionality to be driven by the common or differing aspects of the features model. Figure 4-9 provides an example of an OO1 FMap[1].

Validation of the 001-represented functional model is based upon the syntax and semantics analysis from the FMap analyzer and the execution of the domain model prototype. The FMap analyzer checks to ensure that all parts of the definitions are internally consistent and checks the interfaces for correctness and completeness. The domain model prototype enables the functional model to be executed and validated against known applications by verifying that the prototype exhibits the common functionality of the applications and, by selecting a specific feature(s), verifying that the functionality associated with that feature(s) performs correctly. Section 5 will discuss in further detail the execution of the prototype and the validation process of the functional model.

```
mvmt_data1=schedule_convoy(mvmt_data0)j,cj*7;

        mvmt_model0=moveto:mvmt_model:mvmt_data(mvmt_data0);
      envir0=moveto:environment:mvmt_model(mvmt_model0);
     features0=moveto:features:mvmt_model(mvmt_model0);
    convoy_length=calc_convoy_length(envir0,features0)-op-;
   traversal_time=calc_route_traversal_time(convoy_length,envir0,features0)-op-;
  route_traversal_time=Nat:Rat(traversal_time);
 envir1=determine_schedule(route_traversal_time,features0,envir0)-op-;
mvmt_model1=moveto:mvmt_model:environment(envir1);
mvmt_data1=moveto:mvmt_data:mvmt_model(mvmt_model1);
```

Figure 4-9 Example of an OO1 FMap

---

[1] Please refer to References [001SRM] for an in-depth discussion of the syntax and semantics of the OO1 FMaps.

5 Application of the Domain Model in System Development

The domain model forms the basis for the software development of a system within the defined domain. An executable (or enactable) domain model supports model validation and enhances new systems development within the domain. The executable domain model enables the domain analyst to validate the domain model by parameterizing the model, executing it, and comparing the results against existing applications. The executable domain model can support the user/developer interaction in developing a new system. The executable domain model can serve as a basis for understanding the user's needs and obtaining requirements. Users and developers can identify requirements which already exist as capabilities in the domain model, implement a working model of the system by instantiating a set of existing capabilities, comprehend the resulting behavior and functionality exhibited by the system as a result of the selected capabilities, and define the unprecedented development needed to complete the requirements for the system. This support can also aid in the identification of areas of reuse.

A prototype is described as "an enactable mock-up or model of a software system that enables evaluation of features or functions through user and developer interaction with operational scenarios" [SEI92b]. By automating the domain model, a prototype is essentially being created for the domain model. The following subsection discusses the integration of an automated support tool into the products of a domain analysis and the capabilities of prototyping the domain model. The final two subsections discuss the prototyping capability of the 001 Tool Suite and the use of the prototyping capability of 001 to validate the 001 representation of the domain model.

5.1 The Domain Model and Prototyper Capability

A domain modeling tool can be used in the early stages of the software development process to support end users in specifying requirements for a new system. This tool must provide the three views of the FODA domain model. While the information captured in the model provides enough information to build a system, automatic prototyping gives the user the ability to animate the specification built from a selection of features.

To meet this need for integration, the domain modeling tool must support both domain model creation and model-based prototyping (see Figure 5-1). Using the domain modeling tool, the domain analyst documents the domain model and implements a prototyping capability (labeled Domain Model Prototyper in the figure). The prototyper allows implementation of a working model of the system under development based on a selection of features. The fidelity of the prototype is a function of:

- The completeness of the domain model (how many of the features within the domain have been captured in the model)

- The implementation of code generating capability in the prototyper for those features

As the domain model matures, more features will be captured in the domain model and, as prototyping verifies parts of the model, the prototyping capability will be increased.
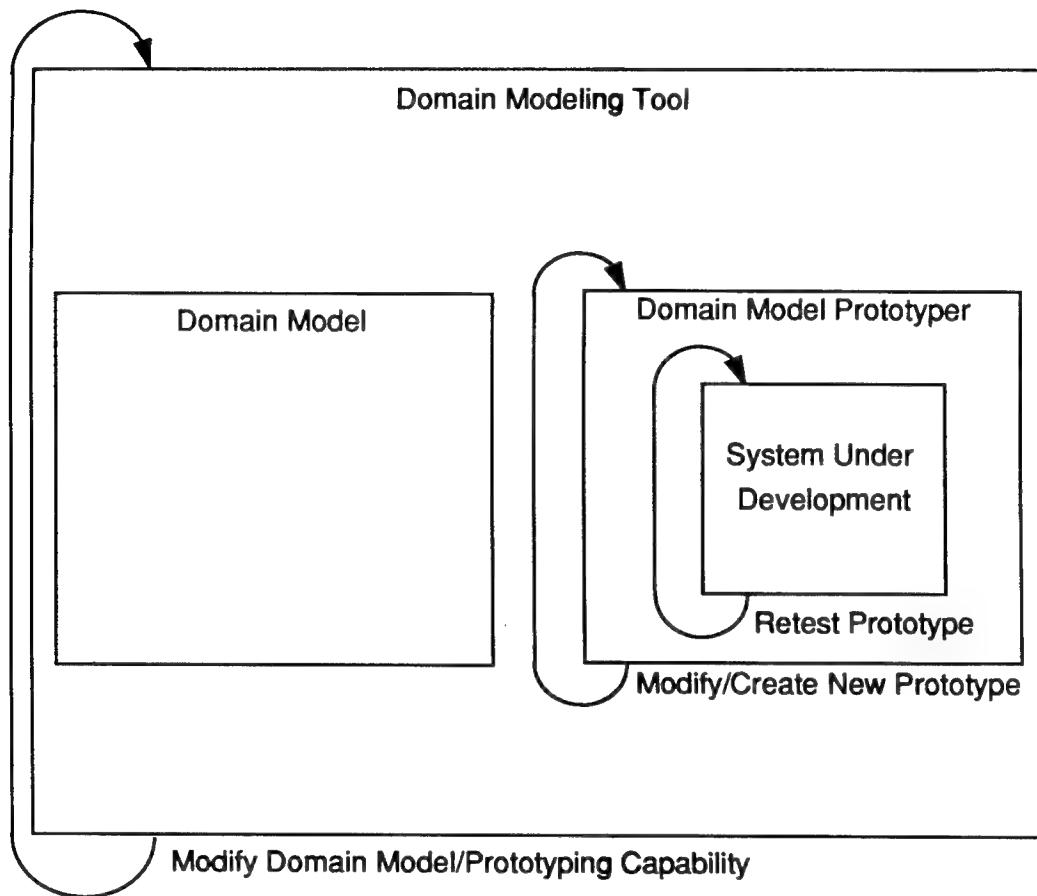
Figure 5-1 Domain Modeling Tool Capability

The loops within the figure show the pattern of development of both the model and the prototyper to validate the model. The outermost loop shows the evolution of the model. As the domain analyst captures more domain information, the model is expanded or changed to reflect new domain information. In parallel with domain model development, the domain analyst will construct a working model of the domain via the prototyper. A selection of features captured in the domain model will be built into the prototyper to allow both validation of the domain model (Does the model capture the common features of systems in the domain?) and prototyping of a system under development.

The loops attached to the prototyper show the ability of the domain modeling tool to test new system capability.

- Given an existing domain model, the prototyper allows the selection of features to implement a prototype for a system under development. The user of the prototyper may change the features selected for the system to build a new or modified version.

- The prototype may be successively tested. Errors may be traced to incorrect selection of features for the system under development or to incorrect implementation of the features within the prototyper.

The prototyper supports both the domain model developer and user:

- The developer gets feedback on the correctness and completeness of the model.

- The user gets a sense of the capabilities of a new system and the effect of selecting alternative or optional features.

The prototyper directly supports the notion of the *binding time* of features, as described in the FODA report:

- *Compile time*: the features implemented in the prototyper that cannot be changed without modifying the domain model or the prototype

- *Load time*: the features that can be changed whenever the prototyper is used to build a new prototype system or to modify an existing system

- *Runtime: the* features that can be selected during execution of the system

The next subsection describes the use of 001 for system prototyping.

5.2 Prototyping the Domain Model

By integrating access to the domain model products (the feature, information, and functional models), the domain modeling tool built using 001 can provide a prototyping capability for system development. An 001 executable model is created by linking the source code automatically generated from the 001 RAT, for each of the Tmap and FMap's definitions. Associated with the linking process, 001 automatically generates separate data and test harnesses around the executable model. These harnesses provide the utility to test and interface with the executable model during execution. The resulting executable represents the prototype of the system (i.e., a convoluted executable of all products of the domain modeling phase).

In the schema of Figure 5-1, the OO1 domain model is constructed using the concepts discussed in Section 4 The OO1 executable model created from the OO1 domain model represents the prototype of the system under development. The OO1-generated test harnesses serve as the domain model prototyper. Retesting of the prototype is performed by re-executing the OO1 executable model with the selection of an alternative set of input parameters (features). Modifications to the domain model/prototyping capabilities are made by modifications

and/or additions to the TMap and FMap definitions. The modification of the prototype (or creation of a new prototype) is accomplished by re-linking the modified domain model.

The OMap editor was used as the default-form user interface for the system during prototyping. The OMap editor is an 001 utility used to create, store, load, modify, and view any complex object made from the TMap. The OMap editor enables the user to enter data before and during execution and review the results afterwards.

The use of the 001 prototyping capability and the OMap editor supports the notions of feature selection and binding time of features when implementing a working model of a system under development. The OMap represents a runtime instance of the objects defined in the TMap. Previously instantiated and stored versions of an OMap may represent the input data (entities and features) for executing the prototype. This stored version of the OMap would represent the input object database and compile time features selected[1]. By editing the OMap prior to execution, entity databases may be altered and load-time features selected. During execution, the OMap editor provides a user interface to communicate with the executing prototype. This permits the user to select runtime features and respond to functional issues surrounding the system being prototyped.

The flexibility permitted in the prototyping process enables any system in the domain to be examined based on the features selected and the resulting behavior and functionality exhibited in the prototype. As the domain model matures, the 001 domain model can grow by expanding the information in the TMap and FMap definitions. The expanded executable model is created by passing the new definitions through the definitions analyzer and RAT and linking this source code into the previous source code. The executable model now includes the new system capabilities for prototyping. Validation of the 001-represented domain model is the topic of the following subsection.

5.3 Validating the Domain Model

Validation of the 001-represented domain model was performed by using the 001-generated data and test harnesses and the prototype of the domain model. The data harness and prototype were used to apply test cases and observe the results. The test harness was used to identify errors if the prototype did not accurately represent the system being modeled.

The domain model was created by representing a *family* of systems in a domain. By parameterizing the functional model, each specific application should be able to be recreated. Therefore, validation of the domain model centers around reproducing known applications through the selection of specific features and issues/decisions during execution of the prototype. Any variation between the predicted and actual results should indicate problems with the description of the system in the model. When variations occur, the 001 test harness was used to identify the discrepancies.

---

[1] Compile time features may also be defined by building the selected features directly into the TMap definition prior to linking the FMap and TMap definitions into the executable model.

When executing a prototype, the 001 test harness automatically calls a local debugger for the host machine and language of the generated source code. The debugger enables the prototype, created from the domain model, to be monitored for errors. If the prototype does not accurately represent the known system being modeled, the debugger can be used to track down the error. Once the error is located, the definition(s) within the model can be edited, passed to the 001 Analyzer, turned into source code by the RAT, and linked back into the executable model. The validation process would continue by executing this new prototype.

6 Conclusions

This report documents the identified need to integrate additional tool support into the FODA method. The tool support should offer an integrated environment for collecting and retrieving the large volume of domain information. The feasibility study stated that [SEI90a, p. 9]:

> *The use of the FODA method in this feasibility study, while successful in explicitly setting forth the capabilities of systems in the domain, is not yet a complete success for the method. The method produces accurate products which describe the domain, but these products have not been used in the implementation of new applications. When this has been done, then the method may be considered a success.*

Therefore, both the process of domain analysis and the process by which the products of domain analysis support software development should benefit from the additional tool support.

This report examined the integration of 001 into the FODA methodology. The effort focused on the ability of 001 to represent the components of the FODA domain model, the integration of the components into a unified representation of the domain model, and the use of the domain model in generating applications.

This report documents the outcome of the 001 Tool Suite's ability to enhance the FODA methodology. This report was not intended to imply that 001 is the one and only solution to the identified need for additional tool support.

6.1 Outcome of 001 Integration

001 provided the ability to represent and integrate all the products in the domain modeling phase of FODA. The 001 TMap proved to be more than adequate in representing the information captured in the features and information models. The 001 FMaps provided all of the capabilities necessary to represent the functional model. By using 001 to represent the domain model rather than the set of manual and independent semi-automated methods used during the feasibility study, the features, information, and functional models can be integrated to form a consistent representation of the domain. All aspects of the domain model are developed under a formal specification language (the 001 AXES). Each component of the domain model can be verified separately or in combination with the other products of the domain modeling phase. The ability to create an executable of the 001-represented domain model extended the usefulness of the model by including a prototyping capability of applications in the domain. Therefore, with 001, both the creation of domain products which model the domain and the software implementation of systems from the domain products were enhanced.

This study began by representing the domain information already captured via previous modeling techniques into an 001 representation. The process of representing the domain model with 001 went very smoothly. The structure of the TMap provided a straightforward, understandable representation of the features and entities (from the information model) of the do-

main. The TMap enabled the entities to be represented at any level of abstraction and decomposition with reuse of identified common structures of decomposed entities. The TMap provides a natural structure for the hierarchical decomposition of features and the facility to establish features as being mandatory, alternative, or optional.

In 001, the FMaps and TMap are integrated. This integration of FMaps and TMap enhanced the understanding of the functionality and behavior associated with specific feature(s) and the entities being affected. FMaps provided the ability to represent the functional and behavioral aspects of the applications within the domain. 001 provides the model developer with the flexibility to develop functionality and behavior common to all the applications in the domain as well as the functionality and behavior only associated with a specific feature(s). Because FMaps and the TMap are integrated together, the FMaps were used to monitor and manipulate the features and entities on the TMap. For example, FMaps were used to establish the link associated with the composition rules of a particular feature.

The ability to generate source code from TMap and FMap definitions and create an executable for the domain model provided the ability to prototype applications in the domain. Prototyping enabled the domain model to be validated against known applications and to represent new applications in the domain. Issues such as binding time of features, feature selection (or parameterization of the functional model), and composition rules could all be examined via the prototype. The prototype serves as a baseline for communication between a systems developer and an end user in establishing requirements for a new application and identifying the areas of reuse or unprecedented development.

There were two areas where the 001 representation did not meet the expectations of a FODA support tool. Firstly, the 001 representation of the functional model lacks a formal portrayal of a "classical functional model" (i.e., a data flow diagram and a state transition diagram) for expressing high-level perspectives of functionality and behavior. This is important for a quick and basic understanding of what the functional model is representing within the domain. Secondly, the representation of features and entities in the TMap would be more complete if 001 offered the ability to "tag" a textual definition to each feature or entity. For example, a form of hypertext-like definition similar to that found in the FMaps utilities would be useful.

Both of these points are important to the FODA methodology. However, it must be pointed out that the primary market for 001 is in software development. 001 is not intended to be a FODA support tool. Therefore, the ability of 001 to represent the domain modeling phase of FODA in no way reflects 001's ability to support software development. In addition, both of these points are currently being addressed by Hamilton Technologies, Inc.

6.2 Future Directions for Tool Support

The future direction for tool support will be twofold. First, the examination of 001 will continue by examining the kinds of user interfaces that 001 can be used with. Currently, a model is executed by selecting the features desired via the OMap editor during execution. However, prim-

itives within the 001 AXES language enable the model developer to access a graphical user interface (such as Motif) within 001 for the selection of features. This graphical interface would create an environment around 001 where the user could select features and enter object data for model execution without an in-depth knowledge of the 001 Tool Suite or the OMap editor. Secondly, an investigation of integrating the FODA domain products into other life cycle or requirements support tools will be conducted. For example, the Software Engineering Information Modeling Project of the SEI is introducing a synthesized technology adapted for application to the requirements engineering domain called AMORE (Advanced Multimedia Organizer for Requirements Elicitation) [SEI93a]. A potential future direction for FODA would be the integration of FODA and AMORE.

References

[001SRM]   The 001™ Tool Suite. *System Reference Manual*, Version 3. Cambridge, Ma.: Hamilton Technologies, Inc., January 1992.

[HAMIL91]   Hamilton, Margaret H.; & Hackler, William R.   "001: A Rapid Development Approach for Rapid Prototyping Based on a System that Supports its Own Life Cycle." *pp. 46-62. Proceedings of the First International Workshop on Rapid System Prototyping.* IEEE Computer Society Press, June 1991.

[HAREL89]   Harel, David, et al. "Statemate: A Working Environment for the Development of Complex Reactive Systems." *IEEE Transactions on Software Engineering* 16, 4 (April 1990): 403-414.

[MURPH90]   Murphy, Erin E. "Software R & D: From an Art to a Science." *IEEE Spectrum* 27, 10 (October 1990): 44-46.

[RUMB91]   Rumbaugh, James, et al. *Object-Oriented Modeling and Design.* Englewood Cliffs, N.J.:, Prentice Hall, 1991.

[SEI90a]   Kang, Kyo C.; Cohen, Sholom G.; Hess, James A.; Novak, William E.; & Peterson, A. Spencer. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1990.

[SEI92a]   Cohen, Sholom G.; Stanley Jr., Jay L.; Peterson, A. Spencer; & Krut Jr., Robert W. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain* (CMU/SEI-91-TR-28, ADA 256590). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.

[SEI92b]   Wood, David P.; & Kang, Kyo C. *A Classification and Bibliography of Software Prototyping* (CMU/SEI-92-TR-13, ADA 258466). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.

[SEI93a]   Christel, Michael G.; & Wood, David P.; & Stevens, Scott M. *AMORE: The Advanced Multimedia Organizer for Requirements Elicitation* (CMU/SEI-93-SR-12). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.

[STA1]   i-Logix, Inc. *Statemate: Support for Large Scale Projects, Version 3.0.* Burlington, Ma.: i-Logix, Inc., January 1990.

[STA2]   i-Logix, Inc. *The Languages of STATEMATE.* Burlington, Ma.: i-Logix, Inc., July 1990.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	

4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-93-TR-11	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-93-188

6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office

6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (city, state, and zip code) HQ ESC/ENS 5 Eglin Street Hanscom AFB, MA 01731-2116

8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003

8c. ADDRESS (city, state, and zip code)) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS.			
	PROGRAM ELEMENT NO 63756E	PROJECT NO. N/A	TASK NO N/A	WORK UNIT NO. N/A

11. TITLE (Include Security Classification)
Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology

12. PERSONAL AUTHOR(S)
Robert W. Krut, Jr.

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) July 1993	15. PAGE COUNT 42 pp.

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	001
			domain analysis
			feature-oriented domain analysis (FODA)

19. ABSTRACT (continue on reverse if necessary and identify by block number)

This report addresses the need for additional tool support for the Feature-Oriented Domain Analysis (FODA) methodology, developed at the Software Engineering Institute (SEI). Previous FODA studes relied on multiple tools to represent the components of a domain model. This report discusses the ability to represent an analyzed domain within the confines of a single support tool. This discussion was based on the transformation of a recently completed domain analysis from a multi-tool, multi-view representation into a single tool which represents the multiple views of a FODA domain model. This report also describes the potential for prototyping of systems using the FODA domain analysis products and the supporting tool.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631	22c. OFFICE SYMBOL ESC/ENS (SEI)

ABSTRACT — continued from page one, block 19

ABSTRACT — continued from page one, block 19